Классы и Объекты

**Класс**— это шаблон для создания объекта, а объект – это реализация класса. Класс определяет структуру и поведение, которые будут совместно использоваться набором объектов. Класс содержит поля и методы, которые называются элементами класса или членами класса.

**На базе класса создается объект:**

**Объект** в программировании — некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением, имеющая определённые свойства (поля) и операции (методы). При создании объекта в переменной мы храним ссылку на объект, который создаем.

**Класс имеет поля с данными:**

**Поле класса** – это переменная любого типа, которая может хранить информацию при создании объекта на базе данного класса.

**Класс имеет методы:**

**Метод класса** — это блок кода, состоящий из ряда инструкций, который можно вызывать по его имени. Он обязательно содержит возвращаемый тип, название, аргументы и тело метода.

**Методы класса могут быть перегружены:**

**Перегрузка методов** – это процесс при котором создается 2 или более методов с одинаковым именем, но принимающее разные параметры. При запуске метода какой метод будет работать зависит от параметров.

**Класс имеет конструктор:**

**Конструктор –** это метод который срабатывает при создании нового объекта, по факту он и возвращает ссылку на объект. Он используется для получения данных при создании и выполнения каких-либо действий на этапе создания объекта.

**Класс может быть абстрактным:**

**Абстрактный класс** – это класс, содержащий методы и поля, которые не могут стать объектом, но могут наследоваться. Абстрактный класс создается с целью создания общей структуры между разными классами наследниками. Он может иметь как обычные методы, так и абстрактные методы с пустым телом.

**Класс может быть образован от интерфейса:**

**Интерфейс** — это конструкция, содержащая набор методов и постоянный полей для будущей реализации схожих по структуре классов.

**Класс, поля класса и методы класса могут иметь ограничения доступа, порождаемые Инкапсуляцией**

**Инкапсуляция** — - механизм ограничения доступа и видимости для скрытия реализации и ограничения возможности взаимодействия (использует модификаторы доступа private, public, protected).

**Класс может наследоваться от другого класса:**

**Наследование** — механизм для перенятия всех полей и методов у класса родителя(используется). Для обращения к классу родителю используется super.

Подкласс или производный класс — класс который наследует

Класс родитель или суперкласс — класс у которого наследуют

**Класс может изменять методы класса родителя благодаря полиморфизму:**

**Полиморфизм -** механизм благодаря которому класс наследник может изменить наследуемый метод от родителя, при этом у родителя метод остается в изначальном виде.

**Поля, методы и вложенные классы могут быть статичными:**

**Для создания статического экземпляра используется ключевое слово static которое обозначает, что какое количество объектов вы не создали бы, их статический экземпляр будет только один и он будет общий для всех объектов. Также к ним можно обращается по имени без создания объекта.**

**Статичные поля** – это поле которое хранит общую информацию для всех порождённых объектов от данного класса. И может быть использовано без создания объектов.

**Статичные методы** – это методы которые можно вызывать без создания объекта класса, но они могут взаимодействовать только с статическими полями класса. Также их можно создавать в интерфейсах. Если это метод класса родителя, то его нельзя переопределить. Абстрактный класс нельзя сделать статичным так-как у него нет теля.

**Статичные вложенные классы** – имеет доступ только к статистическим полям и методам класса верхнего уровня. Он не имеет доступа к нестатистическим полям и методам класса верхнего уровня, но такое решение будет оптимальнее так-как требует меньших затрат в память.

**Поля, методы и классы могут быть финальными:**

**Для создания финального элемента используется слово final которое обозначает что элемент не может быть переопределён в будущем.**

**Финальное поле** – это неизменная переменная предназначенная для создания постоянных значений. Если объявить финальную статическую переменную, то она будет константой.

**Финальный метод** – это метод который нельзя переопределить при наследовании.

**Финальный класс** – это класс от которого нельзя наследовать.

**Методы могут быть рекурсивные:**

**Рекурсия –** это процесс при котором метод вызывает самого себя и при каждом вызове он на шаг ближе к условию остановки.

Класс Object

**Object**– это класс который является корнем иерархии классов. Каждый класс имеет object в качестве суперкласса. Все объекты включая массивы реализуют объекты данного класса.

**Методы(общедоступные):**

**equals(Object obj)** - указывает, является ли какой-либо другой объект "равным" этому.

**getClass()** - возвращает класс среды выполнения этого Object.

**hashCode()** - возвращает значение хэш-кода для объекта.

**notify()** - пробуждает один поток, который ожидает на мониторе этого объекта.

**notifyAll()** - пробуждает все потоки, которые ожидают на мониторе этого объекта.

**toString()** - возвращает строковое представление объекта.

**wait(long timeoutMillis, int nanos)** - заставляет текущий поток ждать, пока он не будет пробужден, обычно путем уведомления или прерывания, или пока не пройдет определенное количество реального времени.

**wait(long timeoutMillis)** - заставляет текущий поток ждать, пока он не будет пробужден, обычно путем уведомления или прерывания, или пока не пройдет определенное количество реального времени.

**wait()** - заставляет текущий поток ждать, пока он не будет пробужден, обычно путем уведомления или прерывания.

**Методы(защищенные):**

**clone()** - создает и возвращает копию этого объекта.

**finalize()** - вызывается сборщиком мусора для объекта, когда сборщик мусора определяет, что ссылок на объект больше нет.

**У классов есть возможность проверки на принадлежность:**

**instanceof –** этоключевоеслово, которое проверяет принадлежит ли конкретный объект конкретному классу.

**Классы можно импортировать:**

**import –** этоключевое слово котороеуказывает компилятору где найти классы на который ссылается объект.

**import static –** эти ключевые слова позволяют импортировать только статические члены классов.

Класс Record

**Record –** это общий базовый класс для всех классов записей языка Java.

**Методы:**

**equals(Object obj)** - указывает, является ли какой-либо другой объект "равным" этому.

**hashCode()** - возвращает значение хэш-кода для записи.

**toString()** - возвращает строковое представление записи.

**Для использования класс Record используется ключевое слово record.**

**Пример:**

public class Human{  
 String name;  
 int age;  
  
 public Human(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 public String getName() { return name;}

public void setName(String name) {this.name = name;}

public int getAge() {return age;}

public void setAge(int age) {this.age = age;}  
  
 @Override  
 public boolean equals(Object o) {  
 if (this == o) return true;  
 if (o == null || getClass() != o.getClass()) return false;  
 Human human = (Human) o;  
 return age == human.age && Objects.*equals*(name, human.name);  
 }  
  
 @Override  
 public int hashCode() {  
 return Objects.*hash*(name, age);  
 }  
  
 @Override  
 public String toString() {  
 return "Human{" +  
 "name='" + name + '\'' +  
 ", age=" + age +  
 '}';  
 }  
}

**Это тоже самое что и:**

public record Human(String name, int age){}

Java Reflection API

**Java Reflection** — это особенный функционал, который позволяет программе получить доступ к приватным частям объектов или поменять поведение некоторых методов классов. Созданный таким образом код будет адаптироваться к входным данным и, например, не будет зависеть от типов, с которыми работает.

Это дает возможность писать код, который со временем будет эволюционировать, то есть не зависеть от текущих имплементаций методов или переменных. Главные преимущества рефлексии — свобода и адаптивность. При необходимости вызвать приватный метод класса можно не переписывать его, а вызвать через Java Reflection. Фактически рефлексия позволяет не следовать написанному коду, вводя новые правила. Можно пойти чуть дальше и начать перехватывать вызовы методов, подменяя их другой логикой, или создать программу, которая будет работать с еще не написанным классом.

**Где используется:**

**При тестировании кода.** Часто бывает нужно проверить корректность работы приватной функции, однако в тесте ее вызвать не получается именно из-за того, что она приватная. Вариантов решения задачи два — сделать ее на время публичной, а потом обратно приватной, или просто вызвать ее в тесте через рефлексию. Второй вариант намного проще и быстрее.

**При написании фреймворков и библиотек.** В популярном Spring Framework рефлексия используется для создания бинов. Во время работы программы Spring Framework собирает данные о классах, помеченных аннотацией `@Component`, и создает для них экземпляры. Это позволяет создавать бины без явного указания их в конфигурационном файле.

**Для поиска и запуска тестов.** Например, так применяет рефлексию библиотека JUnit. Опытные пользователи замечали, что тесты помечены аннотацией `@Test`. Это сделано как раз для того, чтобы во время работы JUnit прошелся по всем классам и запустил всё с этой аннотацией.

**Для сериализации и десериализации объектов.** Например, библиотека Jackson использует рефлексию для сериализации и десериализации объектов в стандарте JSON. Без нее Jackson не смог бы прочитать значения приватных полей и корректно сохранить их в JSON-формате. То же касается и десериализации, когда Jackson должен восстановить значения всех полей, в том числе и приватных, — это было бы невозможно без рефлексии.

Класс Class

**Class** – это класс который расширяет класс Object. Каждый объект имеет класс, которым он был порожден, класс объекта тоже является объектом, который принадлежит и порожден классом Class. У данного класса нет публичного конструктора, а его методы предназначены для получения информации о порождённом классе. Используется для рефлексии.

**Все реализованные интерфейсы:**

Serializable, Constable, TypeDescriptor, TypeDescriptor.OfField<Class<?>>, AnnotatedElement, GenericDeclaration, Type.

**Наследует:**

Object

**Методы:**

**asSubclass(Class<U> clazz)** - преобразует этот Class объект для представления подкласса класса, представленного указанным объектом класса.

**cast(Object obj)** - приводит объект к классу или интерфейсу, представленному этим Class объектом.

**desiredAssertionStatus()** - возвращает статус утверждения, который был бы присвоен этому классу, если бы он был инициализирован во время вызова этого метода.

**forName(String className)** - возвращает Class объект, связанный с классом или интерфейсом, с заданным строковым именем.

**forName(String name, boolean initialize, ClassLoader loader)** - возвращает Class объект, связанный с классом или интерфейсом, с заданным строковым именем, используя заданный загрузчик классов.

**getAnnotatedInterfaces()** - возвращает массив AnnotatedType объектов, которые представляют использование типов для указания суперинтерфейсов сущности, представленной этим Class объектом.

**getAnnotatedSuperclass()** - возвращает AnnotatedType объект, который представляет использование типа для указания суперкласса сущности, представленной этим Class объектом.

**getAnnotation(Class<A> annotationClass)** - возвращает аннотацию этого элемента для указанного типа, если такая аннотация присутствует, иначе null.

**getAnnotations()** - возвращает аннотации, которые присутствуют в этом элементе.

**getAnnotationsByType(Class<A> annotationClass)** - возвращает аннотации, связанные с этим элементом.

**getCanonicalName()** - возвращает каноническое имя базового класса, как определено спецификацией языка Java.

**getClasses()** - возвращает массив, содержащий Class объекты, представляющие все общедоступные классы и интерфейсы, которые являются членами класса, представленного этим Class объектом.

**getClassLoader()** - возвращает загрузчик класса для класса.

**getComponentType()** - возвращает Class, представляющий тип компонента массива.

**getConstructor(Class<?>... parameterTypes) -** возвращает Constructor объект, который отражает указанный общедоступный конструктор класса, представленного этим Class объектом.

**getConstructors() -** возвращает массив, содержащий Constructor объекты, отражающие все общедоступные конструкторы класса, представленного этим Class объектом.

**getDeclaredAnnotation(Class<A> annotationClass)** - возвращает аннотацию этого элемента для указанного типа, если такая аннотация непосредственно присутствует, иначе null.

**getDeclaredAnnotations()** - возвращает аннотации, которые непосредственно присутствуют в этом элементе.

**getDeclaredAnnotationsByType(Class<A> annotationClass) -** возвращает аннотации этого элемента для указанного типа, если такие аннотации либо непосредственно присутствуют, либо косвенно присутствуют.

**getDeclaredClasses()** - возвращает массив Class объектов, отражающих все классы и интерфейсы, объявленные как члены класса, представленного этим Class объектом.

**getDeclaredConstructor(Class<?>... parameterTypes)** - возвращает Constructor объект, который отражает указанный конструктор класса или интерфейса, представленного этим Class объектом.

**getDeclaredConstructors()** - возвращает массив Constructor объектов, отражающий все конструкторы, объявленные классом, представленным этим Class объектом.

**getDeclaredField(String name)** - возвращает Field объект, который отражает указанное объявленное поле класса или интерфейса, представленного этим Class объектом.

**getDeclaredFields()** - возвращает массив Field объектов, отражающих все поля, объявленные классом или интерфейсом, представленным этим Class объектом.

**getDeclaredMethod(String name, Class<?>... parameterTypes)** - Возвращает Method объект, который отражает указанный объявленный метод класса или интерфейса, представленного этим Class объектом.

**getDeclaredMethods() -** возвращает массив, содержащий Method объекты, отражающие все объявленные методы класса или интерфейса, представленные этим Class объектом, включая общедоступные, защищенные, доступ по умолчанию (пакет) и частные методы, но исключая унаследованные методы.

**getDeclaringClass()** - если класс или интерфейс, представленный этим Class объектом, является членом другого класса, возвращает Class объект, представляющий класс, в котором он был объявлен.

**getEnclosingClass()** - возвращает непосредственно заключающий класс базового класса.

**getEnclosingConstructor() -** если этот Class объект представляет локальный или анонимный класс внутри конструктора, возвращает Constructor объект, представляющий непосредственно заключенный конструктор базового класса.

**getEnclosingMethod() -** если этот Class объект представляет локальный или анонимный класс внутри метода, возвращает Method объект, представляющий непосредственно охватывающий метод базового класса.

**getEnumConstants()** - возвращает элементы этого класса enum или null, если этот объект класса не представляет тип enum.

**getField(String name)** - возвращает Field объект, который отражает указанное открытое поле члена класса или интерфейса, представленного этим Class объектом.

**getFields()** - Возвращает массив, содержащий Field объекты, отражающие все доступные общедоступные поля класса или интерфейса, представленные этим Class объектом.

**getGenericInterfaces()** - Возвращает Types, представляющие интерфейсы, непосредственно реализованные классом или интерфейсом, представленным этим объектом.

**getGenericSuperclass() -** возвращает, Type представляющий прямой суперкласс объекта (класс, интерфейс, примитивный тип или void), представленный этим Class.

**getInterfaces()** - определяет интерфейсы, реализованные классом или интерфейсом, представленным этим объектом.

**getMethod(String name, Class<?>... parameterTypes)** Возвращает Method объект, который отражает указанный открытый метод-член класса или интерфейса, представленного этим Class объектом.

**getMethods() -** возвращает массив, содержащий Method объекты, отражающие все общедоступные методы класса или интерфейса, представленные этим Class объектом, включая объявленные классом или интерфейсом и унаследованные от суперклассов и суперинтерфейсов.

**getModifiers()** - возвращает модификаторы языка Java для этого класса или интерфейса, закодированные в целое число.

**getName()** - возвращает имя объекта (класса, интерфейса, класса массива, примитивного типа или void), представленного этим Class объектом, как String.

**getPackage()** - получает пакет для этого класса.

**getProtectionDomain()** - возвращает ProtectionDomain этого класса.

**getResource(String name)** - находит ресурс с заданным именем.

**getResourceAsStream(String name)** - находит ресурс с заданным именем.

**getSigners()** - получает подписчиков этого класса.

**getSimpleName()** - возвращает простое имя базового класса, как указано в исходном коде.

**getSuperclass()** - возвращает, Class представляющий суперкласс объекта (класс, интерфейс, примитивный тип или void), представленный этим Class.

**getTypeName()** - возвращает информативную строку для имени этого типа.

**getTypeParameters()** - возвращает массив TypeVariable объектов, которые представляют переменные типа, объявленные в общем объявлении, представленном этим GenericDeclaration объектом, в порядке объявления.

**isAnnotation()** - возвращает значение true, если этот Class объект представляет тип аннотации.

**isAnnotationPresent(Class<? extends Annotation> annotationClass)** - возвращает значение true, если в этом элементе присутствует аннотация для указанного типа, в противном случае значение false.

**isAnonymousClass()** - возвращает true тогда и только тогда, когда базовый класс является анонимным классом.

**isArray()** - определяет, представляет ли этот Class объект класс массива.

**isAssignableFrom(Class<?> cls)** - определяет, является ли класс или интерфейс, представленный этим Class объектом, либо таким же, либо является суперклассом или суперинтерфейсом класса, или интерфейса, представленных указанным Class параметром.

**isEnum()** - возвращает значение true тогда и только тогда, когда этот класс был объявлен как перечисление в исходном коде.

**isInstance(Object obj)** - определяет, Object совместимо ли указанное назначение с объектом, представленным этим Class.

**isInterface() -** определяет, представляет ли указанный Class объект тип интерфейса.

**isLocalClass()** - возвращает true тогда и только тогда, когда базовый класс является локальным классом.

**isMemberClass() -** возвращает true тогда и только тогда, когда базовый класс является классом-членом.

**isPrimitive() -** определяет, представляет ли указанный Class объект примитивный тип.

**isSynthetic() -** возвращает, true если этот класс является синтетическим классом; возвращает false в противном случае.

**newInstance()** - создает новый экземпляр класса, представленного этим Class объектом.

**toGenericString()** - возвращает строку, описывающую это Class, включая информацию о модификаторах и параметрах типа.

**toString()** - преобразует объект в строку.

Класс AccessibleObject

**AccessibleObject** – это класс который является базовым для классов Field, Method, и Constructor. Он предоставляет возможность пометить рефлексированный объект как подавляющую проверку контроля доступа к языку Java, когда он используется. Это позволяет сложным приложениям с достаточными привилегиями, таким как сериализация объектов Java или другие механизмы сохранения, манипулировать объектами способом, который обычно был бы запрещен.

**Все реализованные интерфейсы:**

AnnotatedElement

**Прямые известные подклассы:**

Executable, Field

**Расширяет:**

Object

**Методы:**

**canAccess​(Object obj)** - проверьте, может ли вызывающий объект получить доступ к этому рефлектированному объекту.

**getAnnotation​(Class<T> annotationClass)** - возвращает аннотацию этого элемента для указанного типа, если такая аннотация присутствует, иначе null.

**getAnnotations()** - возвращает аннотации, которые присутствуют в этом элементе.

**getAnnotationsByType​(Class<T> annotationClass)** - возвращает аннотации, связанные с этим элементом.

**getDeclaredAnnotation​(Class<T> annotationClass)** - возвращает аннотацию этого элемента для указанного типа, если такая аннотация непосредственно присутствует, иначе null.

**getDeclaredAnnotations()** - возвращает аннотации, которые непосредственно присутствуют в этом элементе.

**getDeclaredAnnotationsByType​(Class<T> annotationClass)** - возвращает аннотации этого элемента для указанного типа, если такие аннотации непосредственно присутствуют или косвенно присутствуют.

**isAccessible()** - не рекомендуется. Этот метод устарел, поскольку его название намекает на то, что он проверяет, доступен ли рефлексированный объект, когда он фактически указывает, подавлены ли проверки для контроля доступа к языку Java.

**isAnnotationPresent​(Class<? extends Annotation> annotationClass)** -возвращает значение true, если в этом элементе присутствует аннотация для указанного типа, в противном случае значение false.

**setAccessible​(boolean flag)** - установите accessible флаг для этого рефлексированного объекта на указанное логическое значение.

**setAccessible​(AccessibleObject[] array, boolean flag)** - удобный способ установить accessible флаг для массива рефлексированных объектов с единственной проверкой безопасности (для эффективности).

**trySetAccessible()** - по возможности установите accessible флаг для этого рефлексированного объекта на true.

Интерфейс Member

**Member** —это интерфейс, который отражает идентифицирующую информацию об отдельном элементе (поле или методе) или конструкторе.

**Все известные реализующие классы:**

Constructor, Executable, Field, Method

**Методы:**

**getDeclaringClass()** - возвращает объект класса, представляющий класс или интерфейс, который объявляет член или конструктор, представленный этим членом.

**getModifiers()** - возвращает модификаторы языка Java для элемента или конструктора, представленного этим элементом, в виде целого числа.

**getName()** - возвращает простое имя базового элемента или конструктора, представленного этим элементом.

**isSynthetic()** - возвращает true если этот элемент был введен компилятором; возвращает false в противном случае.

Класс Field

**Field** **–** это класс которыйпредоставляет информацию об отдельном поле класса или интерфейса и динамический доступ к нему**.**

**Все реализованные Интерфейсы:**

AnnotatedElement, Member.

**Расширяет:**

AccessibleObject

**Методы:**

**equals​(Object obj)** - сравнивает это Field для указанного объекта.

**get​(Object obj)** - возвращает значение поля, представленного этим Field, на указанном объекте.

**getAnnotatedType()** - возвращает объект AnnotatedType, который представляет использование типа для указания объявленного типа поля, представленного этим полем.

**getAnnotation​(Class<T> annotationClass)** - возвращает аннотацию этого элемента для указанного типа, если такая аннотация присутствует, иначе null.

**getAnnotationsByType​(Class<T> annotationClass)** - возвращает аннотации, связанные с этим элементом.

**getBoolean​(Object obj)** - возвращает значение статического поля или экземпляра boolean.

**getByte​(Object obj)** - возвращает значение статического поля или экземпляра byte.

**getChar​(Object obj)** - возвращает значение статического поля или поля экземпляра типа char или другого примитивного типа, преобразуемого в type char посредством расширяющего преобразования.

**getDeclaringClass()** - возвращает Class объект, представляющий класс или интерфейс, который объявляет поле, представленное этим Field объектом.

**getDouble​(Object obj)** - возвращает значение статического поля или поля экземпляра типа double или другого примитивного типа, преобразуемого в type double посредством расширяющего преобразования.

**getFloat​(Object obj)** - возвращает значение статического поля или поля экземпляра типа float или другого примитивного типа, преобразуемого в type float посредством расширяющего преобразования.

**getGenericType()** - возвращает Type объект, который представляет объявленный тип для поля, представленного этим Field объектом.

**getInt​(Object obj)** - возвращает значение статического поля или поля экземпляра типа int или другого примитивного типа, преобразуемого в type int посредством расширяющего преобразования.

**getLong​(Object obj)** - возвращает значение статического поля или поля экземпляра типа long или другого примитивного типа, преобразуемого в type long посредством расширяющего преобразования.

**getModifiers()** - возвращает модификаторы языка Java для поля, представленного этим Field объектом, в виде целого числа.

**getName()** - возвращает имя поля, представленного этим Field объектом.

**getShort​(Object obj)** - возвращает значение статического поля или поля экземпляра типа short или другого примитивного типа, преобразуемого в type short посредством расширяющего преобразования.

**getType()** - возвращает Class объект, который идентифицирует объявленный тип для поля, представленного этим Field объектом.

**hashCode() -** возвращает хэш-код для этого Field.

**isEnumConstant()** - возвращает, true если это поле представляет элемент перечисляемого типа; возвращает false в противном случае.

**isSynthetic()** - возвращает, true если это поле является синтетическим полем; возвращает false в противном случае.

**set​(Object obj, Object value)** - присваивает полю, представленному этим Field объектом в указанном аргументе object, указанное новое значение.

**setAccessible​(boolean flag) -** установите accessible флаг для этого отраженного объекта на указанное логическое значение.

**setBoolean​(Object obj, boolean z)** - задает значение поля как boolean для указанного объекта.

**setByte​(Object obj, byte b)** - задает значение поля как byte для указанного объекта.

**setChar​(Object obj, char c)** - задает значение поля как char для указанного объекта.

**setDouble​(Object obj, double d)** - задает значение поля как double для указанного объекта.

**setFloat​(Object obj, float f)** - задает значение поля как float для указанного объекта.

**setInt​(Object obj, int i)** - задает значение поля как int для указанного объекта.

**setLong​(Object obj, long l)** - задает значение поля как long для указанного объекта.

**setShort​(Object obj, short s)** - задает значение поля как short для указанного объекта.

**toGenericString()** - возвращает строку, описывающую это Field, включая ее общий тип.

**toString()** - возвращает строку, описывающую это Field.

Класс Executable

**Executable** – это класс является суперклассом Method и Constructor. Нужен для их общей функциональности.

**Все реализованные интерфейсы:**

AnnotatedElement, GenericDeclaration, Member

**Прямые известные подклассы:**

Constructor, Method

**Расширяет:**

AccessibleObject

**Методы:**

**getAnnotatedExceptionTypes()** - возвращает массив AnnotatedType объекты, которые представляют использование типов для указания объявленных исключений метода / конструктора, представленных этим исполняемым файлом.

**getAnnotatedParameterTypes()** - возвращает массив AnnotatedType объекты, которые представляют использование типов для указания формальных типов параметров метода/ конструктора, представленных этим исполняемым файлом.

**getAnnotatedReceiverType()** - возвращает AnnotatedType объект, представляющий использование типа для указания типа получателя метода / конструктора, представленного этим Executable объект.

**getAnnotatedReturnType()** - возвращает AnnotatedType объект, который представляет использование типа для указания возвращаемого типа метода / конструктора, представленного этим исполняемым файлом.

**getAnnotation​(Class<T> annotationClass)** - возвращает аннотацию этого элемента для указанного типа, если такая аннотация присутствует, иначе null.

**getAnnotationsByType​(Class<T> annotationClass)** - возвращает аннотации, связанные с этим элементом.

**getDeclaringClass()** - возвращает Class объект, представляющий класс или интерфейс, который объявляет исполняемый файл, представленный этим объектом.

**getExceptionTypes()** - возвращает массив Class объектов, которые представляют типы исключений, объявленных для создания базовым исполняемым файлом, представленным этим объектом.

**getGenericExceptionTypes()** - возвращает массив Type объектов, которые представляют исключения, объявленные для создания этим исполняемым объектом.

**getGenericParameterTypes()** - возвращает массив Type объектов, которые представляют типы формальных параметров в порядке объявления исполняемого файла, представленного этим объектом.

**getModifiers()** - возвращает модификаторы языка Java для исполняемого файла, представленного этим объектом.

**getName()** - возвращает имя исполняемого файла, представленного этим объектом.

**getParameterAnnotations()** - возвращает массив массивов Annotations, которые представляют аннотации к формальным параметрам, в порядке объявления, Executable представленных этим объектом.

**getParameterCount()** - возвращает количество формальных параметров (объявленных явно или неявно, или ни тех, ни других) для исполняемого файла, представленного этим объектом.

**getParameters()** - возвращает массив Parameter объектов, которые представляют все параметры базового исполняемого файла, представленного этим объектом.

**getParameterTypes()** - возвращает массив Class объектов, которые представляют типы формальных параметров в порядке объявления исполняемого файла, представленного этим объектом.

**getTypeParameters()** - возвращает массив TypeVariable объектов, которые представляют переменные типа, объявленные в общем объявлении, представленном этим GenericDeclaration объектом, в порядке объявления.

**isSynthetic()** - возвращает, true если этот исполняемый файл является синтетической конструкцией; возвращает false в противном случае.

**isVarArgs()** - возвращает, true если этот исполняемый файл был объявлен для принятия переменного числа аргументов; возвращает false в противном случае.

**toGenericString()** - возвращает строку, описывающую это Executable, включая любые параметры типа.

Класс constructor

**constructor** – это класс который предоставляет информацию об одном конструкторе для класса и доступ к нему. Разрешает выполнение расширяющих преобразований при сопоставлении фактических параметров newInstance() с формальными параметрами базового конструктора, но выдает IllegalArgumentException если бы произошло сужающее преобразование.

**Все реализованные интерфейсы:**

AnnotatedElement, GenericDeclaration, Member

**Расширяет:**

Executable.

**Методы:**

**equals​(Object obj)** - сравнивает это Constructor для указанного объекта.

**getAnnotatedReceiverType()** - возвращает AnnotatedType объект, представляющий использование типа для указания типа получателя метода / конструктора, представленного этим Executable объект.

**getAnnotatedReturnType()** - возвращает AnnotatedType объект, который представляет использование типа для указания возвращаемого типа метода / конструктора, представленного этим исполняемым файлом.

**getAnnotation​(Class<T> annotationClass)** - возвращает аннотацию этого элемента для указанного типа, если такая аннотация является присутствует, в противном случае значение равно нулю.

**getDeclaredAnnotations()** - возвращает аннотации, которые являются присутствует непосредственно на этом элементе.

**getDeclaringClass()** - возвращает Class объект, представляющий класс, который объявляет конструктор, представленный этим объектом.

**getGenericExceptionTypes()** - возвращает массив Type объекты, которые представляют исключения, объявленные для создания этим исполняемым объектом.

**getGenericParameterTypes()** - возвращает массив Type объекты, которые представляют типы формальных параметров, в порядке объявления, исполняемого файла, представленного этим объектом.

**getName()** - возвращает имя этого конструктора в виде строки.

**getParameterAnnotations()** - возвращает массив массивов из Annotation, которые представляют аннотации к формальным параметрам в порядке объявления Executable представлен этим объектом.

**getParameterCount()** - возвращает количество формальных параметров (объявленных явно или неявно, или ни тех, ни других) для исполняемого файла, представленного этим объектом.

**getTypeParameters()** - возвращает массив TypeVariable объекты, которые представляют переменные типа, объявленные общим объявлением, представленным этим GenericDeclaration объект, в порядке объявления.

**hashCode()** - возвращает хэш-код для этого Constructor.

**isSynthetic()** - возвращает true если этот исполняемый файл является синтетической конструкцией; возвращает false в противном случае.

**isVarArgs()** - возвращает true если этот исполняемый файл был объявлен для приема переменного числа аргументов; возвращает false в противном случае.

**newInstance​(Object... initargs)** - использует конструктор, представленный этим Constructor объект для создания и инициализации нового экземпляра класса объявления конструктора с указанными параметрами инициализации.

**setAccessible​(boolean flag)** - установите accessible присвоите этому рефлексированному объекту указанное логическое значение.

**toGenericString()** - возвращает строку, описывающую это Constructor, включая параметры типа.

**toString()** - возвращает строку, описывающую это Constructor.

Класс Method

**Method** – это класс который предоставляет информацию об одном методе класса или интерфейса и доступ к нему.

**Все реализованные интерфейсы:**

AnnotatedElement, GenericDeclaration, Member

**Расширяет:**

Executable

**Методы:**

**equals​(Object obj)** - сравнивает это Method для указанного объекта.

**getAnnotatedReturnType()** - возвращает AnnotatedType объект, который представляет использование типа для указания возвращаемого типа метода / конструктора, представленного этим исполняемым файлом.

**getAnnotation​(Class<T> annotationClass)** - возвращает аннотацию этого элемента для указанного типа, если такая аннотация присутствует, иначе null.

**getDeclaredAnnotations()** - возвращает аннотации, которые непосредственно присутствуют в этом элементе.

**getDeclaringClass()** - возвращает Class объект, представляющий класс или интерфейс, который объявляет метод, представленный этим объектом.

**getDefaultValue()** - возвращает значение по умолчанию для элемента аннотации, представленного этим Method экземпляром.

**getGenericExceptionTypes()** - возвращает массив Type объектов, которые представляют исключения, объявленные для создания этим исполняемым объектом.

**getGenericParameterTypes()** - возвращает массив Type объектов, которые представляют типы формальных параметров в порядке объявления исполняемого файла, представленного этим объектом.

**getGenericReturnType() -** возвращает Type объект, который представляет формальный возвращаемый тип метода, представленного этим Method объектом.

**getName()** - возвращает имя метода, представленного этим Method объектом, в виде String.

**getParameterAnnotations()** - возвращает массив массивов Annotations, которые представляют аннотации к формальным параметрам, в порядке объявления, Executable представленных этим объектом.

**getParameterCount()** - возвращает количество формальных параметров (объявленных явно или неявно, или ни того, ни другого) для исполняемого файла, представленного этим объектом.

**getReturnType()** - возвращает Class объект, который представляет формальный возвращаемый тип метода, представленного этим Method объектом.

**getTypeParameters()** - возвращает массив TypeVariable объектов, которые представляют переменные типа, объявленные в общем объявлении, представленном этим GenericDeclaration объектом, в порядке объявления.

**hashCode()** - возвращает хэш-код для этого Method.

**invoke​(Object obj, Object... args)** - вызывает базовый метод, представленный этим Method объектом, для указанного объекта с указанными параметрами.

**isBridge()** - возвращает, true если этот метод является методом моста; возвращает false в противном случае.

**isDefault()** - возвращает, true если этот метод является методом по умолчанию; возвращает false в противном случае.

**isSynthetic()** - возвращает, true если этот исполняемый файл является синтетической конструкцией; возвращает false в противном случае.

**isVarArgs()** - возвращает, true если этот исполняемый файл был объявлен для принятия переменного числа аргументов; возвращает false в противном случае.

**setAccessible​(boolean flag)** - установите accessible флаг для этого рефлексированного объекта на указанное логическое значение.

**toGenericString()** - возвращает строку, описывающую это Method, включая параметры типа.

**toString()** - возвращает строку, описывающую это Method.

Способы создания объектов

В Java существует четыре различных способа создания объектов:

1. с помощью ключевого слова new.
2. методом newInstance().
3. методом clone().
4. десериализацией объекта.

**Ключевое слово new**

**Ключевое слово new –** это наиболее распространенный способ создания объекта в Java. Ключевое слово new создает экземпляр класса, выделяя память для нового экземпляра указанного типа. После new идет конструктор — специальный метод, отвечающий за создание объекта и инициализацию полей создаваемого объекта.

Object object = new Object();

Object object – создание хранилища.

= - запись в хранилище ссылку, которую получим далее.

new Object() – создание экземпляр класса Object в память.

Object() – конструктор который возвращает ссылку на объект.

**Метод newInstance()**

**newInstance()** – это метод который может возвращать объект определенного класса. Данный метод делится на два вида:

1. Метод newInstance() из API java.lang.Class
2. Метод newInstance() из API java.lang.reflect.Constructor

**newInstance() из Class API** – это метод который не учитывает не каких параметров или аргументов, так что его можно назвать конструктором без параметров. Достаточно создать объект типа Class, передать в него класс и вызвать данный метод.

**newInstance() из Constructor API** – это схожий метод с методом из Class API. Но для создания требуется создать объект Constructor и передать ему конструктор, а после использовать у объекта данный метод.

**Метод clone()**

**clone** – это метод являющийся частью класса Object. Он создает объект класса без вызова какого-либо конструктора класса. Чтобы клонировать метод, соответствующий класс должен иметь реализованный интерфейс Cloneable, который является интерфейсом маркера.

Клонированный объект будет ссылаться на тот же исходный объект через ссылку. Однако клонированный объект будет иметь отдельное назначение памяти. Это означает, что любые изменения, сделанные в объекте по ссылке, не изменят исходный объект. Это связано с тем, что метод clone() создает поверхностную копию объектов.

**десериализация объектов**

**Десериализация объекта** —это процесс извлечения объекта из серии потоков байтов. Сериализация делает обратное. Его основная цель — извлечь сохраненный объект из базы данных/сети обратно в память. Если мы хотим сериализовать или десериализовать объект, нам нужно реализовать интерфейс Serializable (интерфейс маркера). Сначала сериализуем объект по ссылке в текстовый файл. Метод writeObject() запишет поток байтов объекта в текстовый файл. Затем, используя десериализацию объекта, мы извлекаем объект обратно. Точно так же метод readObject() будет считывать объект из потока ввода объекта.